

Chapter 14

Network Flow IV - Applications II

CS 573: Algorithms, Fall 2014

October 14, 2014

14.0.1 Airline Scheduling

14.0.1.1 Airline Scheduling

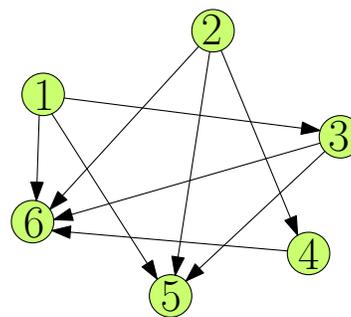
Problem 14.0.1. Given information about flights that an airline needs to provide, generate a profitable schedule.

- (A) Input: detailed information about “legs” of flight.
- (B) \mathcal{F} : set of flights by
- (C) Purpose: find minimum # airplanes needed.

14.0.2 Example

14.0.2.1 (i) a set \mathcal{F} of flights that have to be served, and (ii) the corresponding graph G representing these flights.

- 1: Boston (depart 6 A.M.) - Washington DC (arrive 7 A.M.),
- 2: Urbana (depart 7 A.M.) - Champaign (arrive 8 A.M.)
- 3: Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)
- 4: Urbana (depart 11 A.M.) - San Francisco (arrive 2 P.M.)
- 5: San Francisco (depart 2:15 P.M.) - Seattle (arrive 3:15 P.M.)
- 6: Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.).



(i)

(ii)

14.0.2.2 Flight scheduling...

- (A) Use same airplane for two segments i and j :
 - (a) destination of i is the origin of the segment j ,
 - (b) there is enough time in between the two flights.
- (B) Also, airplane can fly from $\text{dest}(i)$ to $\text{origin}(j)$ (assuming time constraints are satisfied).

Example 14.0.2. As a concrete example, consider the flights:

1. Boston (depart 6 A.M.) - Washington D.C. (arrive 7 A.M.),
2. Washington (depart 8 A.M.) - Los Angeles (arrive 11 A.M.)
3. Las Vegas (depart 5 P.M.) - Seattle (arrive 6 P.M.)

This schedule can be served by a single airplane by adding the leg “Los Angeles (depart 12 noon)- Las Vegas (1 P.M.)” to this schedule.

14.0.2.3 Modeling the problem

- (A) model the feasibility constraints by a graph.
- (B) G : directed graph over flight legs.
- (C) For i and j (legs), $(i \rightarrow j) \in E(G) \iff$ same airplane can serve both i and j .
- (D) G is acyclic.
- (E) Q: Can required legs can be served using only k airplanes?

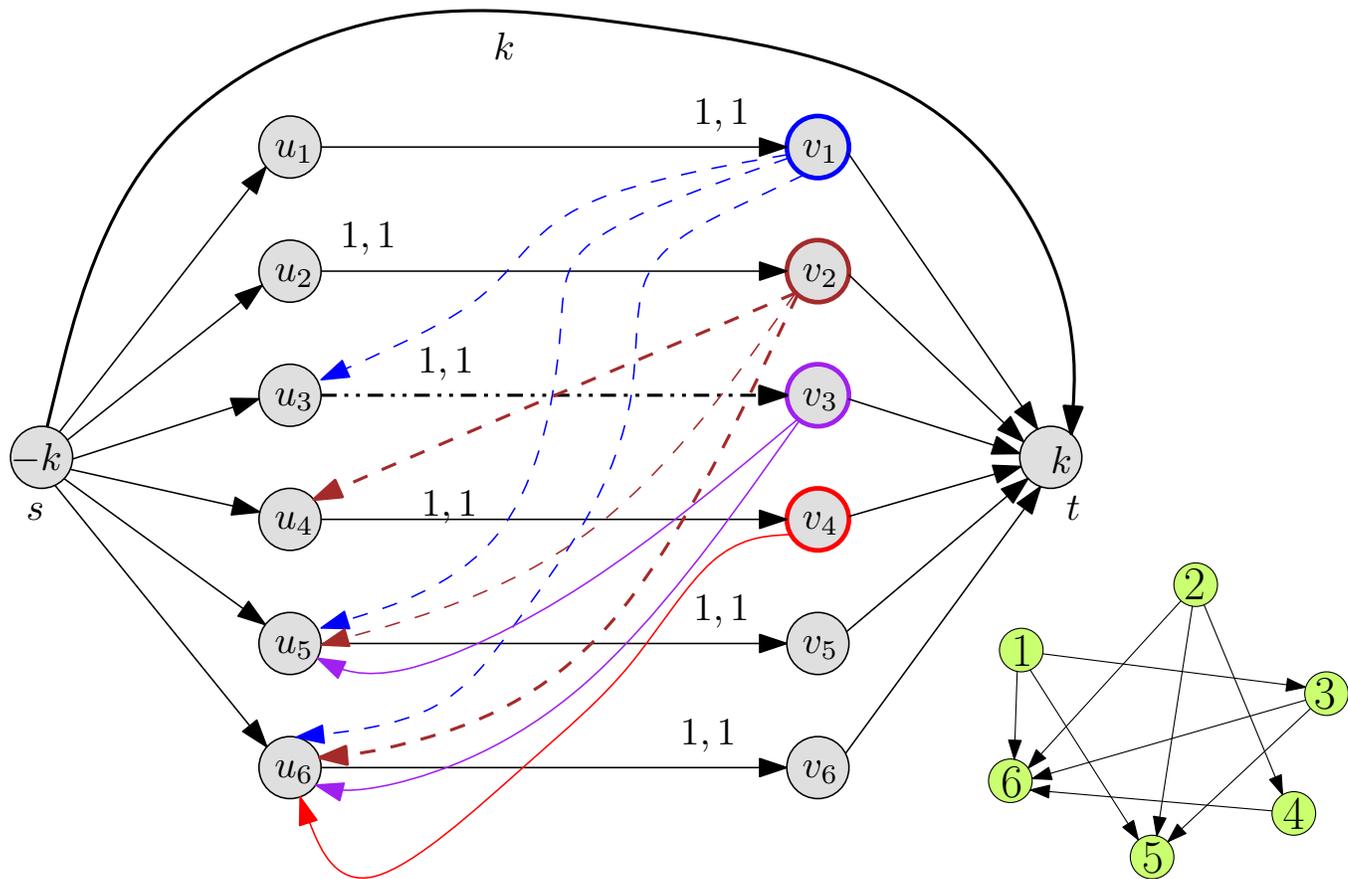
14.0.2.4 Solution

- (A) Reduction to computation of circulation.
- (B) Build graph H .
- (C) \forall leg i , two new vertices $u_i, v_i \in V_H$.
 s : source vertex. t : sink vertex.
- (D) Set demand at t to k , Demand at s to be $-k$.
- (E) Flight must be served: New edge $e_i = (u_i \rightarrow v_i)$, for leg i .
Also $\ell(e_i) = 1$ and $c(e_i) = 1$.
- (F) If same plane can so i and j (i.e., $(i \rightarrow j) \in E(G)$) then add edge $(v_i \rightarrow u_j)$ with capacity 1 to H .
- (G) Since any airplane can start the day with flight i : add an edge $(s \rightarrow u_i)$ with capacity 1 to H , $\forall i$.
- (H) Add edge $(v_j \rightarrow t)$ with capacity 1 to G , $\forall j$.
- (I) Overflow airplanes: “overflow” edge $(s \rightarrow t)$ with capacity k .

Let H denote the resulting graph.

14.0.3 Example of resulting graph

14.0.3.1 The resulting graph H for the instance of airline scheduling show before.



14.0.3.2 Lemma

Lemma 14.0.3. \exists way perform all flights of $\mathcal{F} \leq k$ planes $\iff \exists$ circulation in H .

Proof: (A) Given feasible solution \rightarrow translate into valid circulation.

(B) Given feasible circulation...

(C) ... extract paths from flow.

(D) ... every path is a plane.

14.0.3.3 Extensions and limitations

(A) a lot of other considerations:

- (i) airplanes have to undergo long term maintenance treatments every once in awhile,
- (ii) one needs to allocate crew to these flights,
- (iii) schedule differ between days, and
- (iv) ultimately we interested in maximizing revenue.

(B) Network flow is used in practice, real world problems are complicated, and network flow can capture only a few aspects.

(C) ... a good starting point.

14.1 Image Segmentation

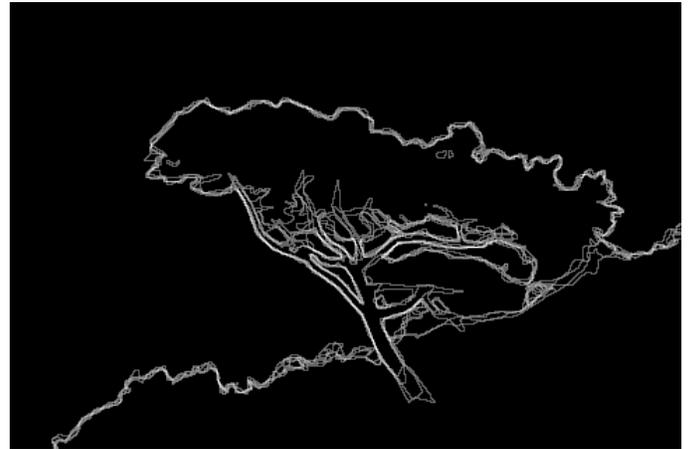
14.1.0.4 Image Segmentation

Input is an image.

Partition image into background and foreground.



(i)

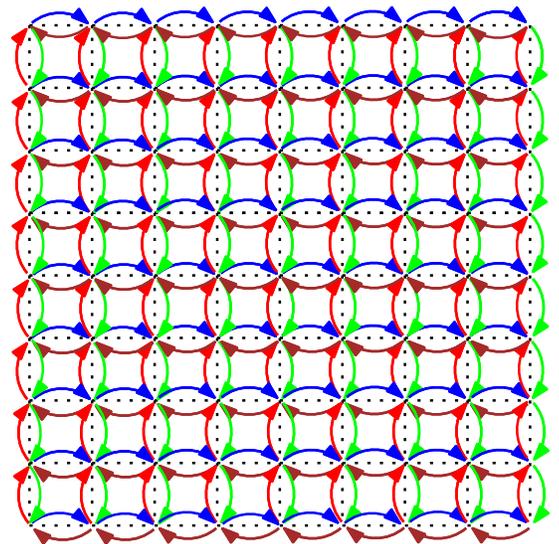


(ii)

The (i) input image, and (ii) a possible segmentation of the image.

14.1.0.5 What is the input...

- (A) Input is a bitmap on a grid.
- (B) Every grid node represents a pixel
- (C) Convert grid into a directed graph G ,
- (D) Input:
 - (i) $N \times N$ bitmap. $G = (V, E)$.
 - (ii) \forall pixel i : foreground value $f_i \geq 0$.
 - (iii) \forall pixel i : background value b_i .
 - (iv) $\forall i, j$ adjacent: separation penalty p_{ij} .
(we assume that $p_{ij} = p_{ji}$)



14.1.0.6 Problem statement

Problem 14.1.1. Given input as above, partition V (the set of pixels) into two disjoint subsets F and B , such that

$$q(F, B) = \sum_{i \in F} f_i + \sum_{i \in B} b_i - \sum_{(i,j) \in E, |F \cap \{i,j\}|=1} p_{ij}.$$

is maximized.

Rewrite $q(F, B)$ as:

$$\begin{aligned} q(F, B) &= \sum_{i \in F} f_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E, |F \cap \{i,j\}|=1} p_{ij} \\ &= \sum_{i \in v} (f_i + b_i) - \left(\sum_{i \in B} f_i + \sum_{j \in F} b_j + \sum_{(i,j) \in E, |F \cap \{i,j\}|=1} p_{ij} \right). \end{aligned}$$

14.1.0.7 Restating problem...

Maximizing:

$$q(F, B) = \sum_{i \in v} (f_i + b_i) - \left(\sum_{i \in B} f_i + \sum_{j \in F} b_j + \sum_{(i,j) \in E, |F \cap \{i,j\}|=1} p_{ij} \right).$$

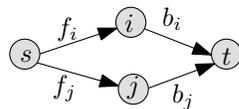
Equivalent to minimizing $u(F, B)$:

$$u(F, B) = \sum_{i \in B} f_i + \sum_{j \in F} b_j + \sum_{(i,j) \in E, |F \cap \{i,j\}|=1} p_{ij}. \quad (14.1)$$

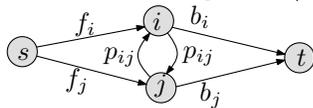
14.1.0.8 Solution continued...

- (A) Compute a minimum cut in a graph. Price = $u(F, B)$.
- (B) A toy example: 
- (C) two possible cuts in the graph:
 - (i) $(\{s, i\}, \{t\})$: price b_i .
 - (ii) $(\{s\}, \{i, t\})$: price f_i .
- (D) Every len 2 path $s \rightsquigarrow t$ forces mincut to choose one of edges.
Mincut “prefers” the edge with lower price.

14.1.0.9 Solution continued...



- (A) Two pixel bitmap:
- (B) Captures background/foreground prices. But... ignores separation penalties...



- (C)
- (D) Price of cut in graph is corresponding value of $u(F, B)$.
- (E) mincut-cut in the resulting graph would corresponds to the required segmentation.

14.1.0.10 Recap...

- (A) Given directed grid graph $G = (V, E)$.
 - (B) s, y : add two special source and sink vertices.
 - (C) $\forall i \in V, :$ add edge $e_i = (s \rightarrow i)$.
 $c(e_i) = f_i$.
 - (D) Add $e'_i = (j \rightarrow t)$ with capacity $c(e'_i) = b_i$.
 - (E) $\forall i, j$ adjacent:
assign the capacity p_{ij} to the edges $(i \rightarrow j)$ and $(j \rightarrow i)$
- H: resulting graph.

14.1.0.11 Solution continues...

By the above discussion:

Lemma 14.1.2. A minimum cut (F, B) in H minimizes $u(F, B)$.

Using the minimum-cut max-flow theorem, we have:

Theorem 14.1.3. One can solve the segmentation problem, in polynomial time, by computing the max flow in the graph H .

14.2 Projection selection

14.2.0.12 Project Selection

- (A) company which can carry out some projects.
- (B) P : set of possible projects.
- (C) $\forall i \in P$: a **revenue** p_i .
- (D) $p_i > 0$ is a profitable project and $p_i < 0$ is a losing project.
- (E) There is dependency between projects.
- (F) $G = (P, E)$: $(i \rightarrow j) \in E \iff j$ is a prerequisite for i .

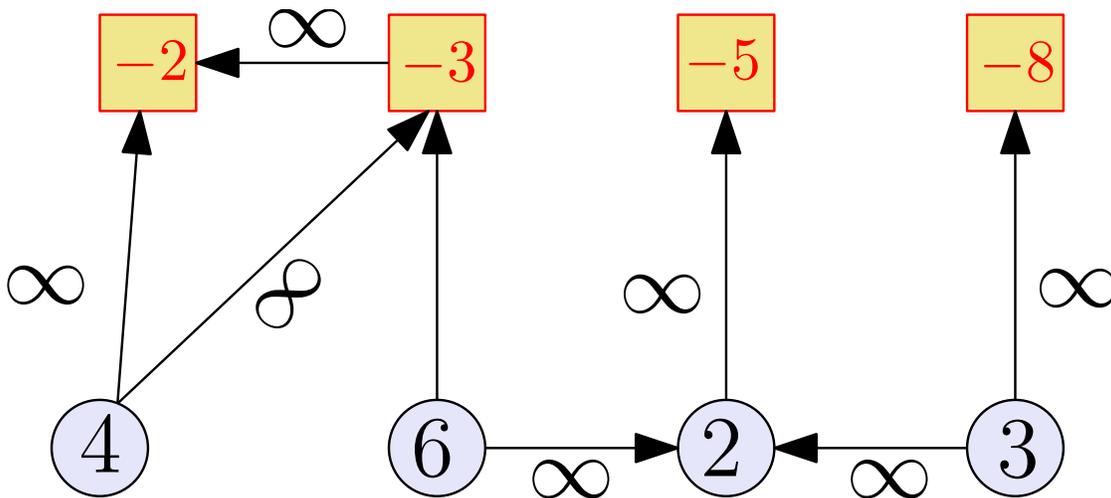
14.2.0.13 Definition

Definition 14.2.1. A set $X \subset P$ is **feasible** if for all $i \in X$, all the prerequisites of i are also in X . Formally, for all $i \in X$, with an edge $(i \rightarrow j) \in E$, we have $j \in X$.

The **profit** associated with a set of projects $X \subseteq P$ is $\text{profit}(X) = \sum_{i \in X} p_i$.

Problem - Project Selection Problem Select a feasible set of projects maximizing the overall profit.

14.2.0.14 Project selection example

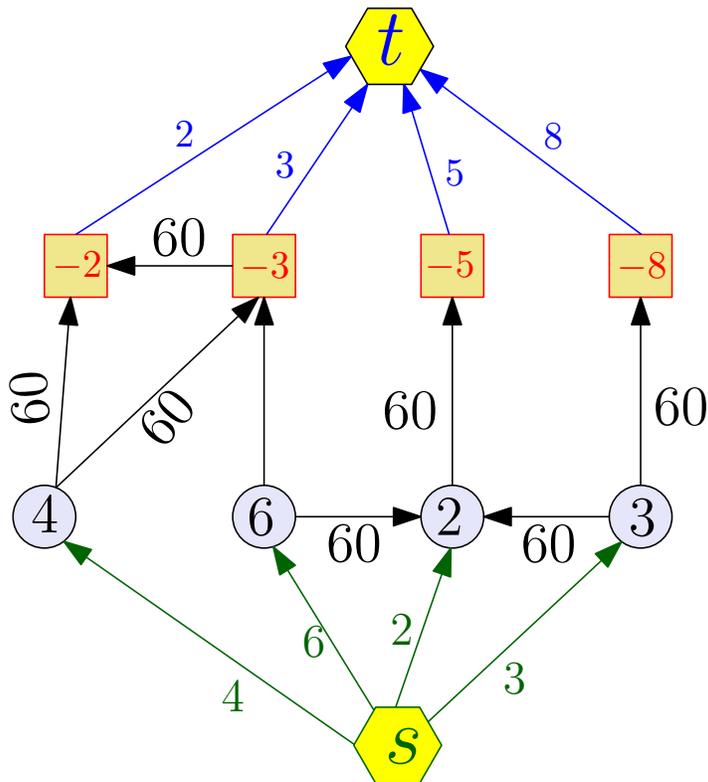


14.2.0.15 The reduction

- (A) Use mincut again.
- (B) Add s and t to G .
- (C) Perform the following modifications:

- (D) $\forall i \in P$ with $p_i > 0$: add edge $e_i = (s \rightarrow i)$.
with $c(e_i) = p_i$.
 - (E) $\forall j \in P$ with $p_j < 0$: add edge $e'_j = (j \rightarrow t)$.
Set $c(e'_j) = -p_j$.
 - (F) $C = \sum_{i \in P, p_i > 0} p_i$: upper bound on profit.
 - (G) Set capacity of all original (dependency) edges in G to $4C$.
- Let H denote the resulting network.

14.2.0.16 Example: Resulting network



14.2.0.17 Solution continued

- (A) $X \subseteq P$: Set of feasible projects.
- (B) $X' = X \cup \{s\}$ and $Y' = (P \setminus X) \cup \{t\}$.
- (C) Consider the s - t cut (X', Y') in H .
- (D) No $E(G)$ is in (X', Y') since X is a feasible set.

14.2.0.18 Lemma

Lemma 14.2.2. $c(X', Y') = C - \sum_{i \in X} p_i = C - \text{profit}(X)$.

Proof

- (A) The edges of H are either:
 - (i) original edges of G ,
 - (ii) emanating from s , and
 - (iii) edges entering t .
- (B) X feasible \implies no edges of type (i) in cut.

(C) Edges entering t contribute:

$$\beta = \sum_{i \in X \text{ and } p_i < 0} -p_i.$$

14.2.0.19 Proof continued

Proof: Edges leaving s contribute:

$$\begin{aligned} \gamma &= \sum_{i \notin X \text{ and } p_i > 0} p_i = \sum_{i \in P, p_i > 0} p_i - \sum_{i \in X \text{ and } p_i > 0} p_i \\ &= C - \sum_{i \in X \text{ and } p_i > 0} p_i, \end{aligned}$$

by the definition of C . The capacity of the cut (X', Y') is

$$\begin{aligned} \beta + \gamma &= \sum_{i \in X \text{ and } p_i < 0} (-p_i) + \left(C - \sum_{i \in X \text{ and } p_i > 0} p_i \right) \\ &= C - \sum_{i \in X} p_i = C - \text{profit}(X), \end{aligned}$$

14.2.0.20 Lemma

Lemma 14.2.3. *If (X', Y') is a cut with capacity at most C in \mathbf{G} , then the set $X = X' \setminus \{s\}$ is a feasible set of projects.*

Namely, cuts (X', Y') of capacity $\leq C$ in \mathbf{H} corresponds one-to-one to feasible sets which are profitable.

Proof: Since $c(X', Y') \leq C$ it must not cut any of the edges of \mathbf{G} , since the price of such an edge is $4C$. As such, X must be a feasible set. ■

14.2.0.21 Summary

- (A) Looking for a feasible set X max: $\text{profit}(X) = \sum_{i \in X} p_i$.
- (B) Corresponds to $X' = X \cup \{s\} \subseteq \mathbf{V}(\mathbf{G})$ minimizes $C - \sum_{i \in X} p_i$.
- (C) = cut capacity (X', Y') .
- (D) Computing min-cut in $\mathbf{H} \equiv$ computing most profitable feasible set projects.

14.2.0.22 Result

Theorem 14.2.4. *If (X', Y') is a minimum cut in \mathbf{H} then $X = X' \setminus \{s\}$ is an optimum solution to the project selection problem. In particular, using network flow the optimal solution can be computed in polynomial time.*

Proof: Indeed, we use network flow to compute the minimum cut in the resulting graph \mathbf{H} . Note, that it is quite possible that the most profitable project is still a net loss. ■

14.3 Baseball Pennant Race

14.3.0.23 Pennant Race

14.3.0.24 Pennant Race: Example

Can Boston win the pennant?

No, because Boston can win at most 91 games.



Example 14.3.1.

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	89	2

14.3.0.25 Another Example

Example 14.3.2.

Team	Won	Left
New York	92	2
Baltimore	91	3
Toronto	91	3
Boston	90	2

Can Boston win the pennant?

Not clear unless we know what the remaining games are!

14.3.0.26 Refining the Example

Can Boston win the pennant? Suppose Boston does

- (A) Boston wins both its games to get 92 wins
- (B) New York must lose both games; now both Baltimore and Toronto have at least 92
- (C) Winner of Baltimore-Toronto game has 93 wins!

14.3.0.27 Abstracting the Problem

Given

- (A) A set of teams S
- (B) For each $x \in S$, the current number of wins w_x
- (C) For any $x, y \in S$, the number of remaining games g_{xy} between x and y
- (D) A team z

Can z win the pennant?

Example 14.3.3.

Team	Won	Left	NY	Bal	Tor	Bos
New York	92	2	–	1	1	0
Baltimore	91	3	1	–	1	1
Toronto	91	3	1	1	–	1
Boston	90	2	0	1	1	–

14.3.0.28 Towards a Reduction

\bar{z} can win the pennant if

(A) \bar{z} wins at least m games

(A) to maximize \bar{z} 's chances we make \bar{z} win all its remaining games and hence $m = w_{\bar{z}} + \sum_{x \in S} g_{x\bar{z}}$

(B) no other team wins more than m games

(A) for each $x, y \in S$ the g_{xy} games between them have to be assigned to either x or y .

(B) each team $x \neq \bar{z}$ can win at most $m - w_x - g_{x\bar{z}}$ remaining games

Is there an assignment of remaining games to teams such that no team $x \neq \bar{z}$ wins more than $m - w_x$ games?

14.3.0.29 Flow Network: The basic gadget

(A) s : source

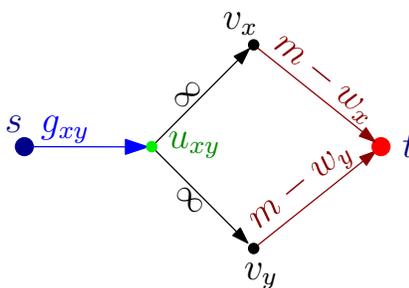
(B) t : sink

(C) x, y : two teams

(D) g_{xy} : number of games remaining between x and y .

(E) w_x : number of points x has.

(F) m : maximum number of points x can win before team of interest is eliminated.

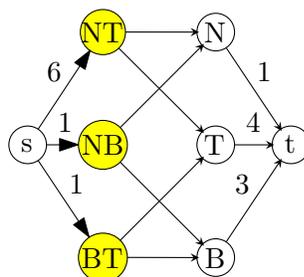


14.3.1 Flow Network: An Example

14.3.1.1 Can Boston win?

Team	Won	Left	NY	Bal	Tor	Bos
New York	90	11	–	1	6	4
Baltimore	88	6	1	–	1	4
Toronto	87	11	6	1	–	4
Boston	79	12	4	4	4	–

(A) $m = 79 + 12 = 91$: Boston can get at most 91 points.



14.3.1.2 Constructing Flow Network

Notations

- (A) S : set of teams,
- (B) w_x wins for each team, and
- (C) g_{xy} games left between x and y .
- (D) m be the maximum number of wins for \bar{z} ,
- (E) and $S' = S \setminus \{\bar{z}\}$.

Reduction Construct the flow network G as follows

- (A) One vertex v_x for each team $x \in S'$, one vertex u_{xy} for each pair of teams x and y in S'
- (B) A new source vertex s and sink t
- (C) Edges (u_{xy}, v_x) and (u_{xy}, v_y) of capacity ∞
- (D) Edges (s, u_{xy}) of capacity g_{xy}
- (E) Edges (v_x, t) of capacity equal $m - w_x$

14.3.1.3 Correctness of reduction

Theorem 14.3.4. G' has a maximum flow of value $g^* = \sum_{x,y \in S'} g_{xy}$ if and only if \bar{z} can win the most number of games (including possibly tie with other teams).

14.3.1.4 Proof of Correctness

Proof: Existence of g^* flow $\Rightarrow \bar{z}$ wins pennant

- (A) An integral flow saturating edges out of s , ensures that each remaining game between x and y is added to win total of either x or y
- (B) Capacity on (v_x, t) edges ensures that no team wins more than m games

Conversely, \bar{z} wins pennant \Rightarrow flow of value g^*

- (A) Scenario determines flow on edges; if x wins k of the games against y , then flow on (u_{xy}, v_x) edge is k and on (u_{xy}, v_y) edge is $g_{xy} - k$

14.3.1.5 Proof that \bar{z} cannot win the pennant

- (A) Suppose \bar{z} cannot win the pennant since $g^* < g$. How do we *prove* to some one *compactly* that \bar{z} cannot win the pennant?
- (B) Show them the min-cut in the reduction flow network!
- (C) See text book for a natural interpretation of the min-cut as a certificate.

14.3.1.6 A compact proof of a team being eliminated

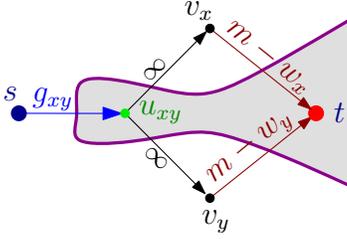
Interestingly, once z is eliminated, we can generate a compact proof of this fact.

14.3.1.7 Helper claim

Claim 14.3.5. For any two teams x and y for which the vertex u_{xy} exists, we have that $u_{xy} \in S$ if and only if both x and y are in \hat{S} .

Proof... $(x \notin \hat{S} \text{ or } y \notin \hat{S}) \implies u_{xy} \notin S$ If x is not in \hat{S} then v_x is in T . But then, if u_{xy} is in S the edge $(u_{xy} \rightarrow v_x)$ is in the cut. However, this edge has infinite capacity, which implies this cut is not a minimum cut (in particular, (S, T) is a cut with capacity smaller than α). As such, in such a case u_{xy} must be in T . This implies that if either x or y are *not* in \hat{S} then it must be that $u_{xy} \in T$. (And as such $u_{xy} \notin S$.)

14.3.1.8 Helper claim proof continued



14.3.1.9 Helper claim proof continued

Proof: $x \in \hat{S}$ and $y \in \hat{S} \implies u_{xy} \in S$ Assume that both x and y are in \hat{S} , then v_x and v_y are in S . We need to prove that $u_{xy} \in S$. If $u_{xy} \in T$ then consider the new cut formed by moving u_{xy} to S . For the new cut (S', T') we have

$$c(S', T') = c(S, T) - c((s \rightarrow u_{xy})).$$

Namely, the cut (S', T') has a lower capacity than the minimum cut (S, T) , which is a contradiction. See figure on the right for this *impossible* cut. We conclude that $u_{xy} \in S$. ■

14.3.1.10 Theorem

Theorem 14.3.6. *Suppose that team z has been eliminated. Then there exists a “proof” of this fact of the following form:*

1. *The team z can finish with at most m wins.*
2. *There is a set of teams $\hat{S} \subset S$ so that $\sum_{x \in \hat{S}} w_x + \sum_{\{x,y\} \subseteq \hat{S}} g_{xy} > m |\hat{S}|$.*

(And hence one of the teams in \hat{S} must end with strictly more than m wins.)

14.3.1.11 Proof

If z is eliminated then the max flow in G has value γ , which is smaller than α . By the minimum-cut max-flow theorem, there exists a minimum cut (S, T) of capacity γ in G , and let $\hat{S} = \{x \mid v_x \in S\}$

The above argumentation implies that edges of the type $(u_{xy} \rightarrow v_x)$ can not be in the cut (S, T) . As such, there are two type of edges in the cut (S, T) : (i) $(v_x \rightarrow t)$, for $x \in \hat{S}$, and (ii) $(s \rightarrow u_{xy})$ where at least one of x or y is not in \hat{S} . As such, the capacity of the cut (S, T) is

14.3.1.12 Proof continued

$$\begin{aligned} c(S, T) &= \sum_{x \in \hat{S}} (m - w_x) + \sum_{\{x,y\} \not\subseteq \hat{S}} g_{xy} \\ &= m |\hat{S}| - \sum_{x \in \hat{S}} w_x + \left(\alpha - \sum_{\{x,y\} \subseteq \hat{S}} g_{xy} \right). \end{aligned}$$

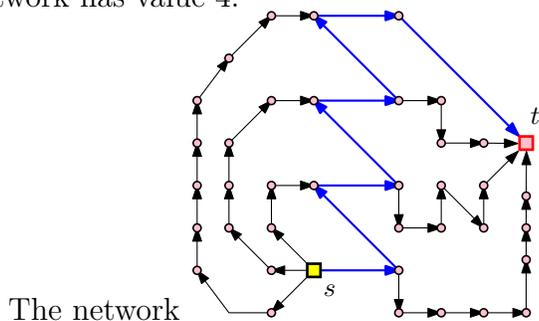
However, $c(S, T) = \gamma < \alpha$, and it follows that

$$m |\hat{S}| - \sum_{x \in \hat{S}} w_x - \sum_{\{x,y\} \subseteq \hat{S}} g_{xy} < \alpha - \alpha = 0.$$

Namely, $\sum_{x \in \hat{S}} w_x + \sum_{\{x,y\} \subseteq \hat{S}} g_{xy} > m |\hat{S}|$, as claimed. ■

14.3.1.13 How much damage can a single path cause?

Consider the following network. All the edges have capacity 1. Clearly the maximum flow in this network has value 4.



Why removing the shortest path might ruin everything

- (A) However... The shortest path between s and t is the blue path.
- (B) And if we remove the shortest path, s and t become disconnected, and the maximum flow drop to 0.